**Day 08 Task Allocation; -**

Creating a full-fledged asset inventory management system in Python with PostgreSQL involves several steps, including setting up the database, creating the necessary tables, and writing Python code to handle CRUD (Create, Read, Update, Delete) operations.

Below is a basic example of a Python program that connects to a PostgreSQL database to manage an asset inventory system for Pan Shops.

**1. Setup PostgreSQL Database**

First, ensure you have PostgreSQL installed and running on your system. You can create a database for the Pan Shop inventory:

**SQL Source Code: -**

**CREATE DATABASE pan\_shop\_inventory;**

### 2. Create Tables in PostgreSQL

You can create a table for the assets with fields such as asset\_id, name, quantity, purchase\_date, and location.

**SQL Source Code: -**

**CREATE TABLE assets (**

**asset\_id SERIAL PRIMARY KEY,**

**name VARCHAR(255) NOT NULL,**

**quantity INT NOT NULL,**

**purchase\_date DATE,**

**location VARCHAR(255)**

**);**

### 3. Python Code to Manage the Asset Inventory

Here's an example of a Python script using the psycopg2 library to interact with the PostgreSQL database:

**Python Source code :-**

**import psycopg2**

**from psycopg2 import sql**

**from datetime import datetime**

**# Establish a connection to the PostgreSQL database**

**def connect\_db():**

**try:**

**conn = psycopg2.connect(**

**dbname="pan\_shop\_inventory",**

**user="your\_username",**

**password="your\_password",**

**host="localhost",**

**port="5432"**

**)**

**return conn**

**except Exception as e:**

**print(f"Error connecting to the database: {e}")**

**return None**

**# Create a new asset**

**def create\_asset(conn, name, quantity, purchase\_date, location):**

**try:**

**with conn.cursor() as cur:**

**cur.execute(**

**"""**

**INSERT INTO assets (name, quantity, purchase\_date, location)**

**VALUES (%s, %s, %s, %s)**

**RETURNING asset\_id;**

**""",**

**(name, quantity, purchase\_date, location)**

**)**

**conn.commit()**

**asset\_id = cur.fetchone()[0]**

**print(f"Asset created with ID: {asset\_id}")**

**except Exception as e:**

**print(f"Error creating asset: {e}")**

**conn.rollback()**

**# Read asset details**

**def read\_asset(conn, asset\_id):**

**try:**

**with conn.cursor() as cur:**

**cur.execute(**

**"SELECT \* FROM assets WHERE asset\_id = %s;", (asset\_id,)**

**)**

**asset = cur.fetchone()**

**if asset:**

**print(f"Asset Details: {asset}")**

**else:**

**print("Asset not found.")**

**except Exception as e:**

**print(f"Error reading asset: {e}")**

**# Update an existing asset**

**def update\_asset(conn, asset\_id, name=None, quantity=None, purchase\_date=None, location=None):**

**try:**

**with conn.cursor() as cur:**

**update\_fields = []**

**update\_values = []**

**if name:**

**update\_fields.append("name = %s")**

**update\_values.append(name)**

**if quantity:**

**update\_fields.append("quantity = %s")**

**update\_values.append(quantity)**

**if purchase\_date:**

**update\_fields.append("purchase\_date = %s")**

**update\_values.append(purchase\_date)**

**if location:**

**update\_fields.append("location = %s")**

**update\_values.append(location)**

**update\_values.append(asset\_id)**

**update\_query = sql.SQL(**

**"UPDATE assets SET {fields} WHERE asset\_id = %s"**

**).format(fields=sql.SQL(", ").join(map(sql.SQL, update\_fields)))**

**cur.execute(update\_query, update\_values)**

**conn.commit()**

**print(f"Asset with ID {asset\_id} updated.")**

**except Exception as e:**

**print(f"Error updating asset: {e}")**

**conn.rollback()**

**# Delete an asset**

**def delete\_asset(conn, asset\_id):**

**try:**

**with conn.cursor() as cur:**

**cur.execute("DELETE FROM assets WHERE asset\_id = %s;", (asset\_id,))**

**conn.commit()**

**print(f"Asset with ID {asset\_id} deleted.")**

**except Exception as e:**

**print(f"Error deleting asset: {e}")**

**conn.rollback()**

**# Main function to test the CRUD operations**

**def main():**

**conn = connect\_db()**

**if conn is None:**

**return**

**# Create a new asset**

**create\_asset(conn, "Pan Masala", 100, datetime(2024, 8, 1), "Shop 1")**

**# Read asset details**

**read\_asset(conn, 1)**

**# Update an asset**

**update\_asset(conn, 1, quantity=150, location="Shop 2")**

**# Delete an asset**

**delete\_asset(conn, 1)**

**# Close the connection**

**conn.close()**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

### Explanation:

* **connect\_db**: Establishes a connection to the PostgreSQL database.
* **create\_asset**: Inserts a new asset into the assets table.
* **read\_asset**: Retrieves and displays asset details based on the asset\_id.
* **update\_asset**: Updates specific fields of an existing asset.
* **delete\_asset**: Deletes an asset from the table based on asset\_id.
* **main**: The main function to test the CRUD operations.

### 4. Requirements

To run this script, you’ll need the psycopg2 library, which can be installed using:

**Bash Source Code:-**

**pip install psycopg2-binary**

### 5. Running the Script

Ensure that your PostgreSQL server is running and the database and table are created as per the steps above. Then, you can run the Python script to manage the asset inventory for the Pan Shops.

This code provides a basic framework. Depending on your specific requirements, you can expand this system with additional features like user authentication, asset categorization, reporting, etc.